Date:	2019-10-30
Author:	Leo Heinz, IPG Solutions Engineering (Germany)
Release No.:	CM-6, CM-7, CM-8

Disclaimer:
This example aims to demonstrate a way to extend the existing CarMaker sensor models to additionally detect traffic lights that were defined in Scenario Editor. Some knowledge of working in C code is expected. This example focuses on the existing CarMaker interface and does not further explain the measures taken to ensure robustness.

The used interface might be subject to change in future CarMaker Releases. As opposed to the examples coming with our CarMaker installation, this one is not maintained or further supported by the IPG Support Team.

How to detect Traffic Lights based on Traffic Sign Sensor?

This is a demonstration of using the Generic model class and of creating an extension based on existing CarMaker models. The result will be a parameterizable model that creates new UAQs similar to what you know from the Traffic Sign Sensor.

Technical Background

The Traffic Sign Sensor compares best to an idealized camera with sign recognition algorithm, relaying the gathered information to a human driver. Such a camera-based recognition could also deliver information of traffic lights within its vision range, which can be used for powertrain economy optimization or higher levels of automated driving. This example shows how to extend the existing ideal sensor information to cover such use cases.

Solution

It was mentioned before that the extension builds on the existing Traffic Sign Sensor. This reduces the number of parameters needed greatly, as the general parameterization of name, position, etc. (compare

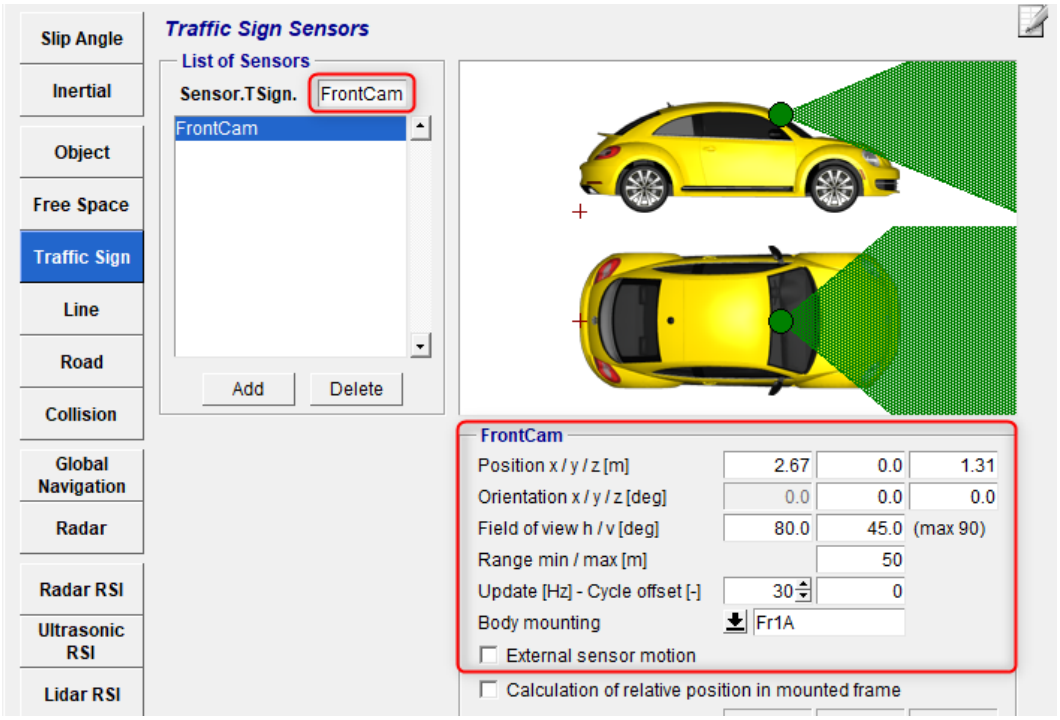

Figure 1: Traffic Sign Sensor in Vehicle Data Set

highlighted area in Figure 1) can be inherited. Aside from that, the sensor needs only little additional information, as shown in Figure 2.
The additional parameter (1) activates the Generic model extension by defining the execution point. It could also be set as Additional TestRun Parameter or in SimParameters. The additional parameters group (2) contains the name of the “parent” Traffic Sign Sensor, the individual parameters will be described with the functional explanation. The extension supports as many Traffic Light Sensors as Traffic Sign Sensors are defined or only a selection of them.

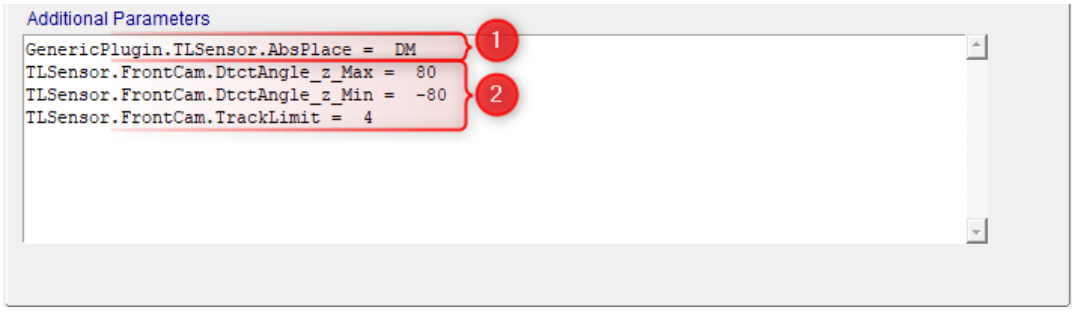

Figure 2: Additional Parameters for Traffic Light Sensor

How do Generic plugin models work?

A model of type “Generic” is used, when the CarMaker environment does not inherently provide a fitting model interface, extending the sensor model functionality being a good example. As there is no distinct interface, a Generic plugin model has no particular position in our main simulation cycle.
The interface is instead defined by directly accessing other signals for reading and writing, be it directly on C code level or via DVA. The model selection and execution point are defined simultaneously by setting an additional parameter “GenericPlugin.<ModelName>.AbsPlace” (as shown in Figure 2). The possible

execution points are similar to Direct Variable Access points. A complete list as well as additional options for Generic models can be found in our Programmer's Guide appendix "Generic Plug-in Models".

How does the model detect Traffic Lights?

For each Traffic Light Sensor the sensor location and orientation in the 3D scene is read from the corresponding Traffic Sign Sensor. The data is available on the C code level by including "Vehicle/Sensor_TSign.h". The corresponding excerpt is shown here:

```
36:   typedef struct tTSignSensor {
37:       tBdySensor  BS;            /* Body sensor                                    */
38:       double      range;          /* max range of beam [m]                  */
39:       double      alpha;          /* beam azimuth [rad]                             */
40:       double      theta;          /* beam elevation [rad]                           */
41:       double      rot_zyx[3];     /* rotation of sensor [rad]               */
42:       double      rot_zyx_ext[3]; /* additional rotation of sensor [rad]            */
43:       double      t_ext[3];       /* additional travel of sensor in FrB [m]  */
44:       double      BS_Pos_0[3];   /* inertial sensor position with ext. motion    */
45:       double      Tr2Fr0[3][3];   /* transformation matrix sensor frame to inertia
46:                                     (FrS -> Fr0)                                  */
47:       double      TimeStamp;      /* time-stamp of sensor calculation       */
48:       int         UpdRate;        /* update rate of sensor signals [Hz]             */
49:       int         nSign;         /* number of detected signs               */
50:       tSign       Sign[ROAD_MAX_TRFSIGNS];
51:   } tTSignSensor;
```

The complete list of traffic lights in the scene, their respective ID, location and state can be obtained from "TrafficLight.h". Here is an excerpt of that section:

```
34:    typedef struct tTrfLightObj {
35:        char       *Name;         /* Traffic light name                  */
36:        tMode      Mode;
37:        tTLState   State;
38:        int        Id;            /* Traffic light Id                    */
39:        int        ObjId;         /* Object Id of all road elements      */
40:        double     Pos_0[3];      /* Global traffic light position in Fr0     */
41:        double     tRemain;       /* Remaining time of the actual light state */
42:    } tTrfLightObj;
43:
44:    typedef struct tTrfLight {
45:        int        nObjs;         /* Number of all traffic lights                */
46:        tTrfLightObj *Objs;              /* Traffic light objects variable handle    */
47:    } tTrfLight;
48:
49:   extern struct tTrfLight TrfLight;
```

The orientation of the Traffic Light is slightly more difficult to obtain. This information is stored with the road information, the global object ID of the traffic light is key to easily access the information. In the demo model, access to the road information is encapsulated in its separate function "int TLSensor_GetTLOrientation (double *orientation, struct tTrfLightObj TrfLight)" with the output storage pointer "orientation" describing the rotation around x, y and z and the traffic light object as defined in the header mentioned above.
Note that the angle obtained from IPGRoad5 is given in degrees and only around the z-axis, as traffic lights are assumed to be positioned parallel to the global z-axis.

With the location and orientation of both the sensors and the traffic lights established, determining the detection can be solved with geometry. For each sensor and each traffic light, first the relative position of the traffic light in the sensor frame (FrS) is established and compared to the vision restrictions of range and field of view (FoV). The second filter is the relative orientation of the traffic light, as those facing away from the sensor could be ignored.

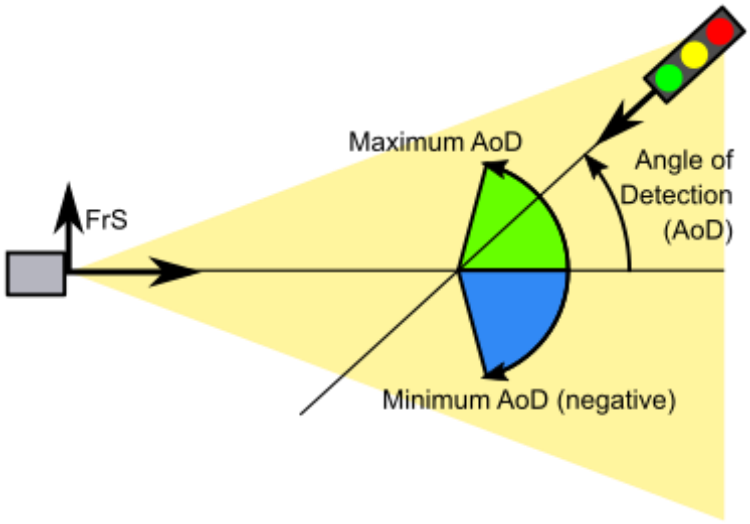

Figure 3: Definition of Angle of Detection

Figure 3 shows a sketch of a traffic light within the sensor range and FoV. The angle of detection is the angle between the orientation of the traffic light and the orientation of the sensor. The use of two parameters “TLSensor.<SName>.DtctAngle_z_Max” and “TLSensor.<SName>.DtctAngle_z_Min” (both in degrees) allows for an asymmetrical orientation filter. The angle of detection is defined counter-clockwise, so the minimum angle of detection is expected as negative value.

The last parameter that also acts as a filter is “TLSensor.<SName>.TrackLimit”. This is the numerical limit of the list of traffic lights that are being monitored. The list is kept in order of polar distance from sensor to traffic light, after the previous detection filters are applied. Several helper functions deal with adding and moving entries in this list.

How can the model be selected?

To use the model extension in your CarMaker project, first make sure you have the project option “Sources / Build Environment” selected in the project creation or update window. Then copy the files “TLSensor.c” and “TLSensor.h” into your projects “src” folder. In “User.c” the header file “TLSensor.h” needs an include call and the function “TLSensor_Register” has to be called in the “User_Register” function as shown in the code snippet below:

```
92:   #include "User.h"
93:   #include "TLSensor.h"
[…]:
213:  int
214:  User_Register (void)
215:  {
216:    TLSensor_Register();
[…]:
221:      return 0;
222:  }
```

For the manual make process also add the source “TLSensor.c” to the Makefile list of objects to be compiled (either directly to line 25 or as “+=” addition as shown in line 26). As shown in the excerpt below, the extension “.O” is used to ensure a precompiled object from the source file is generated.

```
25:   OBJS =                CM_Main.o CM_Vehicle.o User.o
26:   OBJS +=               TLSensor.o
```

Build a new CarMaker executable with these code changes. Keep in mind, that you can have multiple model extensions at the same time. Simply keep their header reference and register function in parallel.

Select a vehicle with a Traffic Sign Sensor or add a Traffic Sign sensor to your vehicle. Then in the “Misc.” tab add the additional parameters described above (compare to Figure 2). The scenario needs to have at least one detectable traffic sign and at least one traffic light for the Traffic Light Sensor to properly initialize. With everything connected, the additional quantities can be found and used as those of any other sensor model, as Figure 4 shows for an example run in CarMaker 8.0.2.

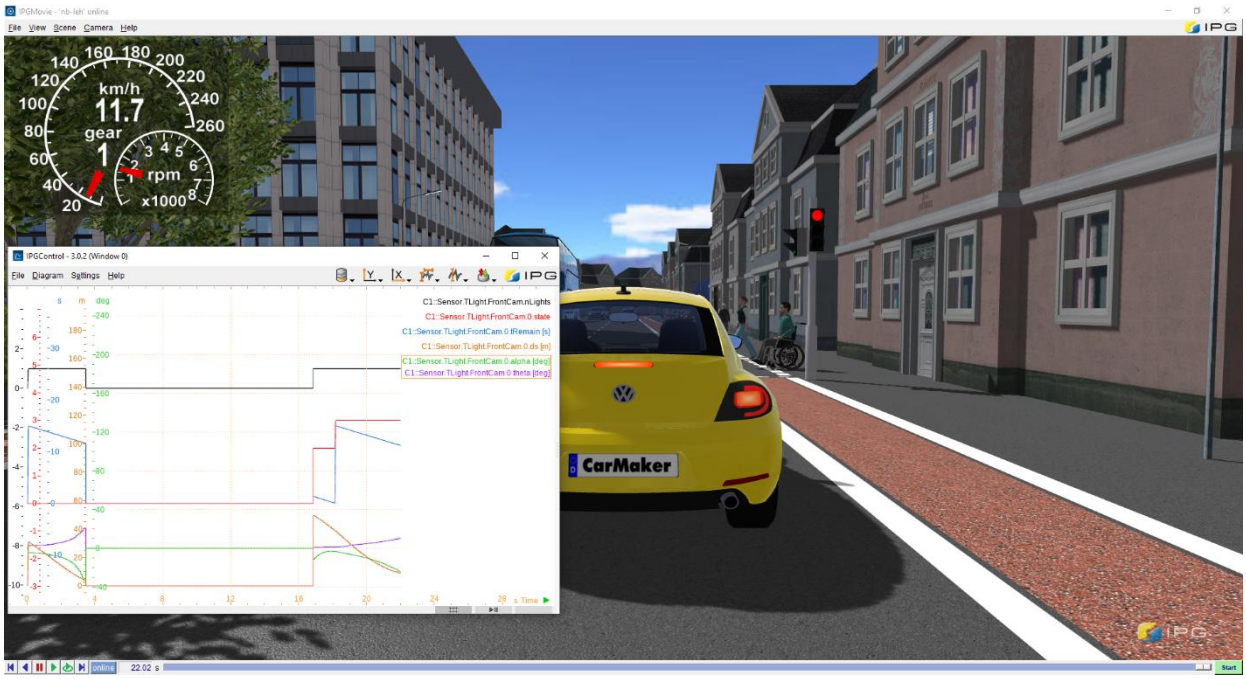

Figure 4: Example scene in IPGMovie with Signals in IPGControl, CarMaker 8.0.2

But does it also work for CarMaker for Simulink?

Like any other C code extension to CarMaker, this model can also be integrated for use with CarMaker for Simulink. To do so, work in the folder “src_cm4sl” of your project instead of “src”. Also set the target Matlab version and its installation directory in the section shown below of the Makefile:

```
24:   MATSUPP_MATVER =        R2016a
25:   MAT_HOME =              C:/Program\ Files/MATLAB/R2016a
```

Compatibility to CarMaker versions 6 and 7

Between major releases the interfaces receive updates for improvement or to connect to new functions and models. Two points need to be highlighted regarding the Traffic Light Sensor extension:

- With CarMaker 7.0 the Paths in IPGRoad received an update, changing the way our road remembers the ego vehicle path. To compensate the initialization in “TLSensor_New” contains a comment with both versions of reading the path for further usage.
- With CarMaker 8.0 the names in enumerations and lists of IPGRoad received an overhaul. To compensate, the filter function “TLSensor_IsValidType” contains a comment with both versions.